EXHIBIT Z

Library of Parameterized Hardware Modules for Floating-Point Arithmetic with An Example Application

A Thesis Presented

by

Pavle Belanović

to

Department of Electrical and Computer Engineering

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in the field of

Electrical Engineering

Northeastern University

Boston, Massachusetts

May 2002

Appendix A

VHDL Entities

Module:

```
parameterized_adder
```

Entity:

```
entity parameterized_adder is
  generic
  (
    bits        : integer           :=  0
  );
  port
  (
    --inputs
    A     : in    std_logic_vector(bits-1 downto 0);
    B     : in    std_logic_vector(bits-1 downto 0);
    CIN   : in    std_logic;
    --outputs
    S     : out   std_logic_vector(bits-1 downto 0) :=   (others=>’0’);
    COUT  : out   std_logic                         :=   ’0’
  );
end parameterized_adder;
```

Module:

```
parameterized_subtractor
```

Entity:

```
entity parameterized_subtractor is
  generic
  (
    bits        : integer           :=  0
  );
  port
  (
    --inputs
    A     : in    std_logic_vector(bits-1 downto 0);
    B     : in    std_logic_vector(bits-1 downto 0);
    --outputs
    O     : out   std_logic_vector(bits-1 downto 0) :=   (others=>’0’)
  );
end parameterized_subtractor;
```

Module:

```
parameterized_multiplier
```

Entity:

```
entity parameterized_multiplier is
  generic
  (
    bits        : integer           :=  0
  );
  port
  (
    --inputs
    A     : in    std_logic_vector(bits-1 downto 0);
    B     : in    std_logic_vector(bits-1 downto 0);
    --outputs
    S     : out   std_logic_vector((2*bits)-1 downto 0) :=  (others=>’0’)
  );
end parameterized_multiplier;
```

Module:

```
parameterized_variable_shifter
```

Entity:

```
entity parameterized_variable_shifter is
  generic
  (
    bits          : integer   :=  0;
    select_bits   : integer   :=  0;
    direction     : std_logic :=  ’0’ --0=right,1=left
  );
  port
  (
    --inputs
    I             : in    std_logic_vector(bits-1 downto 0);
    S             : in    std_logic_vector(select_bits-1 downto 0);
    CLEAR         : in    std_logic;
    --outputs
    O             : out   std_logic_vector(bits-1 downto 0)
  );
end parameterized_variable_shifter;
```

Module:

```
delay_block
```

Entity:

```
entity delay_block is
  generic
  (
    bits  : integer :=  0;
    delay : integer :=  0
  );
  port
  (
    --inputs
    A         : in  std_logic_vector(bits-1 downto 0);
    CLK       : in  std_logic;
    --outputs
    A_DELAYED : out std_logic_vector(bits-1 downto 0) :=  (others=>’0’)
  );
end delay_block;
```

Module:

```
parameterized_absolute_value
```

Entity:

```
entity parameterized_absolute_value is
  generic
  (
    bits  :       integer                            :=  0
  );
  port
  (
    --inputs
    IN1   : in    std_logic_vector(bits-1 downto 0);
    --outputs
    EXC   : out   std_logic                          :=  '0';
    OUT1  : out   std_logic_vector(bits-1 downto 0)   :=  (others=>'0')
  );
end parameterized_absolute_value;
```

Module:

`parameterized_priority_encoder`

Entity:

```
entity parameterized_priority_encoder is
  generic
  (
    man_bits    : integer :=  0;
    shift_bits  : integer :=  0
  );
  port
  (
    --inputs
    MAN_IN        : in    std_logic_vector(man_bits-1 downto 0);
    --outputs
    SHIFT         : out   std_logic_vector(shift_bits-1 downto 0) :=  (others=>'0');
    EXCEPTION_OUT : out std_logic                                 :=  '0'
  );
end parameterized_priority_encoder;
```

Module:

`parameterized_mux`

Entity:

```
entity parameterized_mux is
  generic
  (
    bits        : integer           :=  0
  );
  port
  (
    --inputs
    A     : in    std_logic_vector(bits-1 downto 0);
    B     : in    std_logic_vector(bits-1 downto 0);
    S     : in    std_logic;
    --outputs
    O     : out   std_logic_vector(bits-1 downto 0) := (others=>'0')
  );
end parameterized_mux;
```

Module:

`parameterized_comparator`

Entity:

```
entity parameterized_comparator is
  generic
  (
    bits        : integer           :=  0
  );
  port
  (
    --inputs
    A           : in    std_logic_vector(bits-1 downto 0);
    B           : in    std_logic_vector(bits-1 downto 0);
    --outputs
    A_GT_B      : out   std_logic   :=  ’0’;
    A_EQ_B      : out   std_logic   :=  ’0’;
    A_LT_B      : out   std_logic   :=  ’0’
  );
end parameterized_comparator;
```

Module:

`denorm`

Entity:

```
entity denorm is
  generic
  (
    exp_bits    : integer :=  0;
    man_bits    : integer :=  0
  );
  port
  (
    --inputs
    IN1           : in    std_logic_vector(exp_bits+man_bits downto 0);
    READY         : in    std_logic;
    EXCEPTION_IN  : in    std_logic;
    --outputs
    OUT1          : out   std_logic_vector(exp_bits+man_bits+1 downto 0)  :=  (others=>’0’);
    DONE          : out   std_logic                                       :=  ’0’;
    EXCEPTION_OUT : out   std_logic                                       :=  ’0’
  );
end denorm;
```

Module:

`rnd_norm`

Entity:

```
entity rnd_norm is
  generic
  (
    exp_bits      : integer :=  0;
    man_bits_in   : integer :=  0;
    man_bits_out  : integer :=  0
  );
  port
  (
    --inputs
    IN1           : in    std_logic_vector((exp_bits+man_bits_in) downto 0);
    READY         : in    std_logic;
    CLK           : in    std_logic;
    ROUND         : in    std_logic;
    EXCEPTION_IN  : in    std_logic;
    --outputs
    OUT1          : out   std_logic_vector((exp_bits+man_bits_out) downto 0)  :=  (others=>’0’);
    DONE          : out   std_logic                                           :=  ’0’;
    EXCEPTION_OUT : out   std_logic                                           :=  ’0’
  );
end rnd_norm;
```

Module:

`fp_add`

Entity:

```
entity fp_add is
  generic
  (
    exp_bits      : integer :=  0;
    man_bits      : integer :=  0
  );
  port
  (
    --inputs
    OP1           : in    std_logic_vector(man_bits+exp_bits downto 0);
    OP2           : in    std_logic_vector(man_bits+exp_bits downto 0);
    READY         : in    std_logic;
    EXCEPTION_IN  : in    std_logic;
    CLK           : in    std_logic;
    --outputs
    RESULT        : out   std_logic_vector(man_bits+exp_bits+1 downto 0)  :=  (others=>’0’);
    EXCEPTION_OUT : out   std_logic                                       :=  ’0’;
    DONE          : out   std_logic                                       :=  ’0’
  );
end fp_add;
```

Module:

`fp_sub`

Entity:

```
entity fp_sub is
  generic
  (
    exp_bits      : integer :=  0;
    man_bits      : integer :=  0
  );
  port
  (
    --inputs
    OP1           : in    std_logic_vector(man_bits+exp_bits downto 0);
    OP2           : in    std_logic_vector(man_bits+exp_bits downto 0);
    READY         : in    std_logic;
    EXCEPTION_IN  : in    std_logic;
    CLK           : in    std_logic;
    --outputs
    RESULT        : out   std_logic_vector(man_bits+exp_bits+1 downto 0)  :=  (others=>’0’);
    EXCEPTION_OUT : out   std_logic                                       :=  ’0’;
    DONE          : out   std_logic                                       :=  ’0’
  );
end fp_sub;
```

Module:

`fp_mul`

Entity:

```
entity fp_mul is
  generic
  (
    exp_bits      : integer :=  0;
    man_bits      : integer :=  0
  );
  port
  (
    --inputs
    OP1           : in    std_logic_vector(exp_bits+man_bits downto 0);
    OP2           : in    std_logic_vector(exp_bits+man_bits downto 0);
    READY         : in    std_logic;
    EXCEPTION_IN  : in    std_logic;
    CLK           : in    std_logic;
    --outputs
    RESULT        : out   std_logic_vector(exp_bits+(2*man_bits) downto 0)  :=  (others=>’0’);
    EXCEPTION_OUT : out   std_logic                                         :=  ’0’;
    DONE          : out   std_logic                                         :=  ’0’
  );
end entity;
```

Module:

`fix2float`

Entity:

```
entity fix2float is
  generic
  (
    fix_bits  : integer :=  0;
    exp_bits  : integer :=  0;
    man_bits  : integer :=  0
  );
  port
  (
    --inputs
    FIXED         : in  std_logic_vector(fix_bits-1 downto 0);
    ROUND         : in  std_logic;
    EXCEPTION_IN  : in  std_logic;
    CLK           : in  std_logic;
    READY         : in  std_logic;
    --outputs
    FLOAT         : out std_logic_vector(exp_bits+man_bits downto 0)  :=  (others=>’0’);
    EXCEPTION_OUT : out std_logic                                     :=  ’0’;
    DONE          : out std_logic                                     :=  ’0’
  );
end fix2float;
```

Module:

`float2fix`

Entity:

```
entity float2fix is
  generic
  (
    fix_bits  : integer :=  0;
    exp_bits  : integer :=  0;
    man_bits  : integer :=  0
  );
  port
  (
    --inputs
    FLOAT         : in  std_logic_vector(exp_bits+man_bits downto 0);
    ROUND         : in  std_logic;
    EXCEPTION_IN  : in  std_logic;
    CLK           : in  std_logic;
    READY         : in  std_logic;
    --outputs
    FIXED         : out std_logic_vector(fix_bits-1 downto 0)         :=  (others=>’0’);
    EXCEPTION_OUT : out std_logic                                     :=  ’0’;
    DONE          : out std_logic                                     :=  ’0’
  );
end float2fix;
```

Appendix B

VHDL Description of the IEEE Single Precision Adder

```
--======================================================--
--                      LIBRARIES                       --
--======================================================--
-- IEEE Libraries --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- float
library PEX_Lib;
use PEX_Lib.float_pkg.all;
----------------------------------------------------------
--           IEEE Single Precision Adder                --
----------------------------------------------------------
entity single_precision_adder is
  port
  (
    --inputs
    IN1           : in    std_logic_vector(31 downto 0);
    IN2           : in    std_logic_vector(31 downto 0);
    READY         : in    std_logic;
    EXCEPTION_IN  : in    std_logic;
    ROUND         : in    std_logic;
    CLK           : in    std_logic;
    --outputs
    OUT1          : out   std_logic_vector(31 downto 0) :=  (others=>’0’);
    EXCEPTION_OUT : out   std_logic                     :=  ’0’;
    DONE          : out   std_logic                     :=  ’0’
  );
end single_precision_adder;
----------------------------------------------------------
--           IEEE Single Precision Adder                --
----------------------------------------------------------
architecture single_precision_adder_arch of single_precision_adder is
  signal  rd1       : std_logic                     :=  ’0’;
  signal  rd2       : std_logic                     :=  ’0’;
  signal  rd3       : std_logic                     :=  ’0’;
  signal  rd4       : std_logic                     :=  ’0’;
  signal  exc1      : std_logic                     :=  ’0’;
  signal  exc2      : std_logic                     :=  ’0’;
  signal  exc3      : std_logic                     :=  ’0’;
  signal  exc4      : std_logic                     :=  ’0’;
  signal  rnd1      : std_logic                     :=  ’0’;
  signal  rnd2      : std_logic                     :=  ’0’;
  signal  rnd3      : std_logic                     :=  ’0’;
  signal  rnd4      : std_logic                     :=  ’0’;
  signal  op1       : std_logic_vector(32 downto 0) :=  (others=>’0’);
  signal  op2       : std_logic_vector(32 downto 0) :=  (others=>’0’);
  signal  sum       : std_logic_vector(33 downto 0) :=  (others=>’0’);
begin
  --instances of components
  denorm1: denorm
    generic map
    (
      exp_bits      =>  8,
      man_bits      =>  23
    )
    port map
    (
      --inputs
      IN1           =>  IN1,
      READY         =>  READY,
      EXCEPTION_IN  =>  EXCEPTION_IN,
      --outputs
      OUT1          =>  op1,
      DONE          =>  rd1,
      EXCEPTION_OUT =>  exc1
    );
  denorm2: denorm
    generic map
    (
      exp_bits      =>  8,
      man_bits      =>  23
    )
    port map
    (
      --inputs
      IN1           =>  IN2,
      READY         =>  READY,
      EXCEPTION_IN  =>  EXCEPTION_IN,
      --outputs
      OUT1          =>  op2,
      DONE          =>  rd2,
      EXCEPTION_OUT =>  exc2
    );
  adder: fp_add
    generic map
    (
      exp_bits      =>  8,
      man_bits      =>  24
    )
    port map
    (
      --inputs
      OP1           =>  op1,
      OP2           =>  op2,
      READY         =>  rd3,
      EXCEPTION_IN  =>  exc3,
      CLK           =>  CLK,
      --outputs
      RESULT        =>  sum,
      EXCEPTION_OUT =>  exc4,
      DONE          =>  rd4
    );
  rnd_norm1: rnd_norm
    generic map
    (
      exp_bits      =>  8,
      man_bits_in   =>  25,
      man_bits_out  =>  23
    )
    port map
    (
      --inputs
```